



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт информационных технологий  
Кафедра вычислительной техники

## КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»  
(наименование дисциплины)

Тема курсовой работы К\_6 Моделирование работы банкомата  
(наименование темы)

Студент группы ИКБО-29-22 Азиз Матиулла  
(учебная группа) (Фамилия Имя Отчество) (подпись студента)

Руководитель курсовой работы доцент Лозовский В.В.  
(Должность, звание, ученая степень) (подпись руководителя)

Консультант ст.преп. Асадова Ю.С.  
(Должность, звание, ученая степень) (подпись консультанта)

Работа представлена к защите «20» мая 2023 г.

Допущен к защите «20» мая 2023 г.

ХОРОШО

Москва 2023 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт информационных технологий

Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой

Подпись

Платонова О.В.

ФИО

«21» февраля 2023 г.

#### ЗАДАНИЕ

На выполнение курсовой работы

по дисциплине «Объектно-ориентированное программирование»

Студент Азиз Матиулла Группа ИКБО-29-22

Тема К\_6 Моделирование работы банкомата

Исходные данные:

1. Описания исходной иерархии дерева объектов.
  2. Описание схемы взаимодействия объектов.
  3. Множество команд для управления функционированием моделируемой системы.
- Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Построение версий программ.
2. Построение и работа с деревом иерархии объектов.
3. Взаимодействия объектов посредством интерфейса сигналов и обработчиков.
4. Блок-схемы алгоритмов.
5. Управление функционированием моделируемой системы

Срок представления к защите курсовой работы: до «20» мая 2023 г.

Задание на курсовую работу выдал

(Лозовский В.В.)

ФИО консультанта

Подпись

«21» февраля 2023 г.

Задание на курсовую работу получил

(Азиз М.)

ФИО исполнителя

Подпись

«21» февраля 2023 г.

Москва 2023 г.

## ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент Азиз Матиулла группа ИКБО-29-22  
(ФИО студента) (Группа)

Характеристика курсовой работы

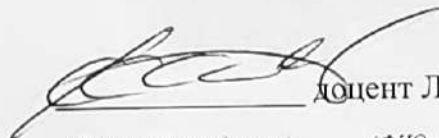
Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	✓		
2. Соответствие курсовой работы заданию	✓		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	✓		
4. Полнота выполнения всех пунктов задания	✓		
5. Логичность и системность содержания курсовой работы	✓		
6. Отсутствие фактических грубых ошибок	✓		

Замечаний:

*Исходные данные корректны, в работе отсутствуют фактические грубые ошибки, работа выполнена качественно, рекомендуется к защите.*

Рекомендуемая оценка:

*хорошо*



доцент Лозовский В.В.

(Подпись руководителя)

(ФИО руководителя)

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 Постановка задачи .....	5
2 Метод решения .....	12
3 Описание алгоритмов .....	14
4 Блок-схемы алгоритмов .....	23
ЗАКЛЮЧЕНИЕ .....	36
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	37
Приложение 1. Код программы .....	38
Приложение 2. Тестирование .....	54

## ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Механизм сигналов и обработчиков является одним из способов организации взаимодействия между объектами в программировании. Он позволяет установить связь между сигналами, которые генерируются объектом, и обработчиками, которые реагируют на эти сигналы. Вместе с передачей сигнала также передаются определенные данные, что делает механизм сигналов и обработчиков мощным инструментом для реализации сложных схем взаимодействия объектов.

В данной работе мы рассмотрим задачу организации взаимосвязи объектов посредством механизма сигналов и обработчиков. Мы опишем алгоритм и метод решения этой задачи, а также представим пример кода, демонстрирующего работу данного механизма. Основная цель работы - исследовать и объяснить принципы работы механизма сигналов и обработчиков, а также продемонстрировать его применение на практике.

В ходе работы мы описываем три основных метода, необходимых для организации взаимосвязи по механизму сигналов и обработчиков: метод установки связи, метод удаления связи и метод выдачи сигнала. Мы предлагаем использование указателей на методы сигнала и обработчика, что позволяет гибко устанавливать и разрывать связи между объектами. Также мы предлагаем использовать структуру для

хранения установленных связей и вектор для управления ими.

## 1 Постановка задачи

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

- установления связи между сигналом текущего объекта и обработчиком целевого объекта;
- удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
- выдачи сигнала от текущего объекта с передачей строковой переменной. Включенный объект может выдать или обработать сигнал.

Методу установки связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу удаления (разрыва) связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу выдачи сигнала передать указатель на метод сигнала и строковую переменную. В данном методе реализовать алгоритм:

- Если текущий объект отключен, то выход, иначе к пункту 2;
- Вызов метода сигнала с передачей строковой переменной по ссылке;
- Цикл по всем связям сигнал-обработчик текущего объекта:
  - Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то проверить готовность целевого

объекта. Если целевой объект готов, то вызвать метод обработчика целевого объекта указанной в связи и передать в качестве аргумента строковую переменную по значению.

- Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать параметризированное макроопределение препроцессора.

В базовый класс добавить метод определения абсолютной пути до текущего объекта. Этот метод возвращает абсолютный путь текущего объекта.

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы. Если при построении дерева иерархии возникает ситуация дуближа имен среди починенных у текущего головного объекта, то новый объект не создается.

Система содержит объекты шести классов с номерами: 1, 2, 3, 4, 5, 6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика.

Каждый метод сигнала с новой строки выводит:

Signal from «абсолютная координата объекта»

Каждый метод сигнала добавляет переданной по параметру строке текста номер класса принадлежности текущего объекта по форме:

«пробел» (class: «номер класса»)

Каждый метод обработчика с новой строки выводит:

Signal to «абсолютная координата объекта» Text: «переданная строка»

Моделировать работу системы, которая выполняет следующие команды с параметрами:

- EMIT «координата объекта» «текст» – выдает сигнал от заданного по координате объекта;
- SET\_CONNECT «координата объекта, выдающего сигнал» «координата целевого объекта» – устанавливает связь;

- DELETE\_CONNECT «координата объекта, выдающего сигнал» «координата целевого объекта» – удаляет связь;
- SET\_CONDITION «координата объекта» «значение состояния» – устанавливает состояние объекта;
- END – завершает функционирование системы (выполнение программы).

Реализовать алгоритм работы системы:

- в методе построения системы:
  - построение дерева иерархии объектов согласно вводу;
  - ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
- в методе отработки системы:
  - привести все объекты в состоянии готовности;
  - цикл до признака завершения ввода:
    - ввод наименования объекта и текста сообщения;
    - вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.
  - конец цикла.

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы. Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.



## 1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве иерархии.  
Структура данных для ввода согласно изложенному в версии № 3 курсовой работы.

После ввода состава дерева иерархии построчно вводится:

«координата объекта, выдающего сигнал» «координата целевого объекта»

Ввод информации для построения связей завершается строкой, которая содержит:

«end\_of\_connections»

В методе запуска (отработки) системы построчно вводятся множество команд в производном порядке:

- EMIT «координата объекта» «текст» – выдать сигнал от заданного по координате объекта;
- SET\_CONNECT «координата объекта, выдающего сигнал» «координата целевого объекта» – установка связи;
- DELETE\_CONNECT «координата объекта, выдающего сигнал» «координата целевого объекта» – удаление связи;
- SET\_CONDITION «координата объекта» «значение состояния» – установка состояния объекта;
- END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

Если координата объекта задана некорректно, то соответствующая операция не выполняется и с новой строки выдается сообщение об ошибке.

Если не найден объект по координате:

Object «координата объекта» not found

Если не найден целевой объект по координате:

Handler object «координата целевого объекта» not found

### Пример ввода:

```
appls_root
/ object_s1 3
/ object_s2 2
/object_s2 object_s4 4
/ object_s13 5
/object_s2 object_s6 6
/object_s1 object_s7 2
endtree
/object_s2/object_s4 /object_s2/object_s6
/object_s2 /object_s1/object_s7
/ /object_s2/object_s4
/object_s2/object_s4 /
end_of_connections
EMIT /object_s2/object_s4 Send message 1
EMIT /object_s2/object_s4 Send message 2
EMIT /object_s2/object_s4 Send message 3
EMIT /object_s1 Send message 4
END
```

## 1.2 Описание выходных данных

### Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

Signal from «абсолютная координата объекта»

Если отработал метод обработчика:

Signal to «абсолютная координата объекта» Text: «переданная строка»

### Пример вывода:

```
Object tree
appls_root
  object_s1
    object_s7
  object_s2
    object_s4
    object_s6
  object_s13
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text: Send message 1 (class: 4)
Signal to / Text: Send message 1 (class: 4)
Signal from /object_s2/object_s4
```

Signal to /object\_s2/object\_s6 Text: Send message 2 (class: 4)  
Signal to / Text: Send message 2 (class: 4)  
Signal from /object\_s2/object\_s4  
Signal to /object\_s2/object\_s6 Text: Send message 3 (class: 4)  
Signal to / Text: Send message 3 (class: 4)  
Signal from /object\_s1

## 2 Метод решения

Для решения задача было добавлено/изменено:

Класс cl\_base:

- Свойства/поля: нет изменений
- Методы:
  - void set\_connect(TYPE\_SIGNAL p\_signal, cl\_base\* p\_object, TYPE\_HANDLER p\_ob\_handler) - создание связи между объектами, модификатор доступа public
  - void delete\_connect(Указатель на метод Signal класса Base, Указатель на объект класса Base, Указатель на метод handler класса Base), удаление связи между объектами, модификатор доступа public
  - void emit\_signal (TYPE\_SIGNAL p\_signal, string& s\_command)-Выдача сигнала , модификатор доступа publicstring get\_address() получения указателя на объект по его абсолютному адресу в дереве иерархии объектов, модификатор доступа public

Класс cl\_application:

- Свойства/поля: нет изменений
- Методы:
  - void signal\_f (string s\_command)- метод Вывода сигнала, модификатор доступа public
  - void handler\_f(string s\_command)- метод обработчика сигнала модификатор доступа public

Иерархия наследования отображена в таблице 1.

*Таблица 1 – Иерархия наследования классов*

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс в иерархии наследования, содержит основной функционал	
		cl_application	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6
		cl_6	public		7
2	cl_application			класс-приложение	
3	cl_2			класс, наследуемый от базового	
4	cl_3			класс, наследуемый от базового	
5	cl_4			класс, наследуемый от базового	
6	cl_5			класс, наследуемый от базового	
7	cl_6			класс, наследуемый от базового	

### 3 Описание алгоритмов

Согласно этапам разработки [3,4], после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

#### 3.1 Алгоритм функции main

Функционал: основная функция программы.

Параметры: отсутствуют.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_application класса cl_application с помощью конструктора с параметром nullptr	2
2		вызов метода build_tree_objects() объекта ob_cl_application	3
3		вызов метода exes_app() объекта ob_cl_application	∅

#### 3.2 Алгоритм метода signal\_f класса cl\_application

Функционал: метод сигнала метод сигнала методы классы cl\_1, cl\_2, cl\_3, cl\_4, cl\_5, cl\_6 аналогичны класса cl\_application.

Параметры: string& msg.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода *signal\_f* класса *cl\_application*

№	Предикат	Действия	№ перехода
1		вывод символа переноса строки, строки "Signal from " и возвращенного значения метода <i>get_way()</i>	2
2		прибавление строки " (class:1)" к переменной <i>msg</i>	∅

### 3.3 Алгоритм метода *handler\_f* класса *cl\_application*

Функционал: метод обработчика, методы классы *cl\_1*, *cl\_2*, *cl\_3*, *cl\_4*, *cl\_5*, *cl\_6* аналогичны класса *cl\_application*.

Параметры: *string& msg*.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода *handler\_f* класса *cl\_application*

№	Предикат	Действия	№ перехода
1		вывод символа переноса строки, строки "Signal to ", возвращенного значения метода <i>get_way()</i> , строки "Text: " и параметра <i>msg</i>	∅

### 3.4 Алгоритм метода *set\_connect* класса *cl\_base*

Функционал: установление связи между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: *TYPE\_SIGNAL p\_signal*, *cl\_base\*p\_object*, *TYPE\_HANDLER p\_handler*.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода *set\_connect* класса *cl\_base*

№	Предикат	Действия	№ перехода
1		создание указателя <i>p_value</i> на структуру <i>o_sh</i>	2
2		инициализация целочисленной переменной <i>i = 0</i>	3
3	<i>i</i> меньше размера вектора <i>connects</i>		4
			5
4	<i>p_signal</i> <i>i</i> -го элемента вектора <i>connects</i> равно параметру <i>p_signal</i> и <i>p_cl_base</i> <i>i</i> -го элемента вектора <i>connects</i> равно параметру <i>p_object</i> и <i>p_handler</i> <i>i</i> -го элемента вектора <i>connects</i> равно параметру <i>p_handler</i>		∅
		увеличение <i>i</i> на 1	5
5		создание объекта <i>p_value</i> структуры <i>o_sh</i>	6
6		поле <i>p_signal</i> объекта <i>p_value</i> равно параметру <i>p_signal</i>	7
7		поле <i>p_cl_base</i> объекта <i>p_value</i> равно параметру <i>p_object</i>	8
8		поле <i>p_handler</i> объекта <i>p_value</i> равно параметру <i>p_handler</i>	9
9		вызов метода <i>push_bask</i> с параметром <i>p_value</i> для вектора <i>connects</i>	∅



### 3.5 Алгоритм метода delete\_connect класса cl\_base

Функционал: удаление (разрыв) связи между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE\_SIGNAL p\_signal, cl\_base\*p\_object, TYPE\_HANDLER p\_handler.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода delete\_connect класса cl\_base

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной i = 0	2
2	i меньше размера вектора connects		3
			∅
3	p_signal i-го элемента вектора connects равно параметру p_signal и p_cl_base i-го элемента вектора connects равно параметру p_object и p_handler i-го элемента вектора connects равно параметру p_handler	вызов метода erase вектора connects с параметром begin вектора connects + i	4
			4
4		увеличение i на 1	2

### 3.6 Алгоритм метода emit\_signal класса cl\_base

Функционал: выдача сигнала от текущего объекта с передачей строковой переменной.

Параметры: TYPE\_SIGNAL p\_signal, string& msg.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода *emit\_signal* класса *cl\_base*

№	Предикат	Действия	№ перехода
1	поле <i>ready</i> равно 0		∅
		вызов метода <i>p_signal</i> с передачей переменной <i>msg</i> по ссылке	2
2	в векторе <i>connects</i> есть объекты		3
			∅
3	поле <i>p_signal</i> текущего объекта равно параметру <i>p_signal</i> и поле <i>p_cl_base</i> имеет поле <i>ready</i> не равное 0	инициализация указателя на базовый класс <i>p_target</i> полем <i>p_cl_base</i>	4
			6
4		инициализация указателя <i>p_handler</i> на метод <i>TYPE_HANDLER</i> полем <i>p_handler</i>	5
5		вызов метода <i>p_handler</i> с передачей переменной <i>msg</i> по ссылке	6
6		переход к следующему объекту	2

### 3.7 Алгоритм метода *build\_tree\_objects\_4* класса *cl\_application*

Функционал: метод построения исходного дерева иерархии объектов (конструирование моделируемой системы).

Параметры: отсутствуют.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода *build\_tree\_objects\_4* класса *cl\_application*

№	Предикат	Действия	№ перехода
1		инициализация указателя на родителя <i>base = this</i>	2
2		инициализация указателя на ребенка <i>base_sub = nullptr</i>	3
3		объявление строковых переменных <i>s_subordinate_name</i> , <i>s_head_name</i> , <i>way</i>	4
4		ввод переменной <i>s_head_name</i>	5

Продолжение таблицы 8

№	Предикат	Действия	№ перехода
5		вызов метода set_s_object_name с параметром s_head_name текущего объекта	6
6		объявление целочисленной переменной i_class	7
7		ввод переменной way	8
8	way не равно "endtree"	ввод переменной s_subordinate_name	9
			18
9		ввод переменной i_class	10
10		приравнивание base возвращенного значения метода get_point с параметром way	11
11	возвращенное значение метода get_subordinate_objects с параметром s_subordinate_name не равно nullptr		12
			17
12	i_class равно 2	создание объекта класса cl_2	17
			13
13	i_class равно 3	создание объекта класса cl_3	17
			14
14	i_class равно 4	создание объекта класса cl_4	17
			15
15	i_class равно 5	создание объекта класса cl_5	17
			16
16	i_class равно 6	создание объекта класса cl_6	17
			17
17		ввод переменной way	8
18		объявление строковых переменных signal_way, handler_way	19
19		объявление указателей на базовый класс p_signal, p_handler	20
20		ввод переменной signal_way	21
21	signal_way не равно "end_of_connections"	ввод переменной handler_way	22
			∅
22		p_signal равно возвращенному значению метода get_point с параметром signal_way	23
23		p_handler равно возвращенному значению метода get_point с параметром handler_way	24
24	p_signal равно nullptr	вывод переноса строки, строки "Object", переменной signal_way и строки "not found"	27
			25

Продолжение таблицы 8

№	Предикат	Действия	№ перехода
25	p_handler равно nullptr	вывод переноса строки, строки "Handler object", переменной handler_way и строки "not found"	27
			26
26		вызов метода set_connect с передачей параметров возвращенное значение метода get_signal с параметром cl_num указателя p_signal, переменная p_handler и возвращенное значение метода get_handler с параметром cl_num указателя p_handler от указателя p_signal	27
27		ввод переменной signal_way	21

### 3.8 Алгоритм метода exes\_app\_4 класса cl\_application

Функционал: метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Параметры: отсутствуют.

Возвращаемое значение: целое.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода exes\_app\_4 класса cl\_application

№	Предикат	Действия	№ перехода
1		вывод строки "Object tree"	2
2		вызов метода print_from_current_3()	3
3		вызов метода set_ready_for_tree с параметром 1	4
4		объявление строковых переменных command, way, signal_way, handler_way, msg	5
5		объявление указателей на базовый класс p_signal, p_handler, p_obj	6
6		объявление целочисленной переменной ready	7
7		ввод переменной command	8
8	command не равно "END"		9
			∅
9	command равно "EMIT"	ввод переменной signal_way	10
			15

Продолжение таблицы 9

№	Предикат	Действия	№ перехода
10		p_signal равно возвращенному значению get_point с параметром signal_way	11
11		считывание строки в переменную msg	12
12	p_signal равно nullptr	вывод переноса строки, строки "Object", переменной signal_way и строки "not found"	13
			14
13		ввод переменной command	8
14		вызов метода emit_signal с параметрами возвращенное значение метода get_signal с параметром cl_num от указателя p_signal и msg от p_signal	38
15	command равно "SET_CONNECT"	ввод переменной signal_way	16
			24
16		ввод переменной handler_way	17
17		p_signal равно возвращенному значению get_point с параметром signal_way	18
18		p_handler равно возвращенному значению get_point с параметром handler_way	19
19	p_signal равно nullptr	вывод переноса строки, строки "Object", переменной signal_way и строки "not found"	20
			21
20		ввод переменной command	8
21	p_handler равно nullptr	вывод переноса строки, строки "Handler object", переменной handler_way и строки "not found"	22
			23
22		ввод переменной command	8
23		вызов метода set_connect с передачей параметров возвращенное значение метода get_signal с параметром cl_num указателя p_signal, переменная p_handler и возвращенное значение метода get_handler с параметром cl_num указателя p_handler от указателя p_signal	38
24	command равно "DELETE_CONNECT"	ввод переменной signal_way	25
			33
25		ввод переменной handler_way	26
26		p_signal равно возвращенному значению get_point с параметром signal_way	27
27		p_handler равно возвращенному значению get_point с параметром handler_way	28
28	p_signal равно nullptr	вывод переноса строки, строки "Object", переменной signal_way и строки "not found"	29
			30
29		ввод переменной command	8

Продолжение таблицы 9

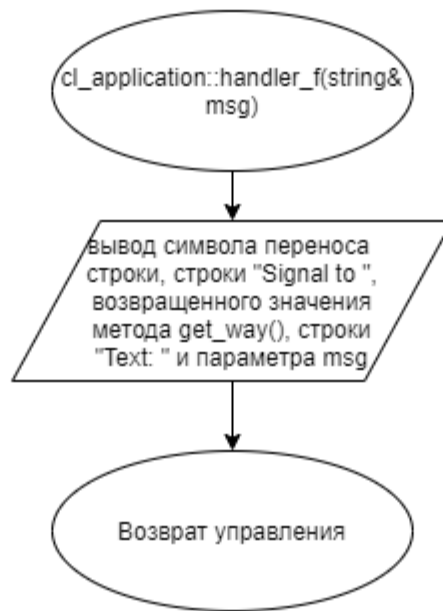
№	Предикат	Действия	№ перехода
30	p_handler равно nullptr	вывод переноса строки, строки "Handler object", переменной handler_way и строки "not found"	31
			32
31		ввод переменной command	8
32		вызов метода delete_connect с передачей параметров возвращенное значение метода get_signal с параметром cl_num указателя p_signal, переменная p_handler и возвращенное значение метода get_handler с параметром cl_num указателя p_handler от указателя p_signal	38
33	command равно "SET_CONDITION"	ввод переменной way	34
			38
34		ввод переменной ready	35
35		p_obj равно возвращенному значению метода get_point с параметром way	36
36	p_obj равно nullptr	вывод переноса строки, строки "Object", переменной way и строки "not found"	37
			38
37		ввод переменной command	8
38		вызов метода set_reay с параметром ready от p_obj	39
39		ввод переменной command	8

## 4 Блок-схемы алгоритмов

Представим описание алгоритмов в графическом виде на рисунках 1-13.



Рисунок 1 – Блок-схема алгоритма



**Рисунок 2 – Блок-схема алгоритма**



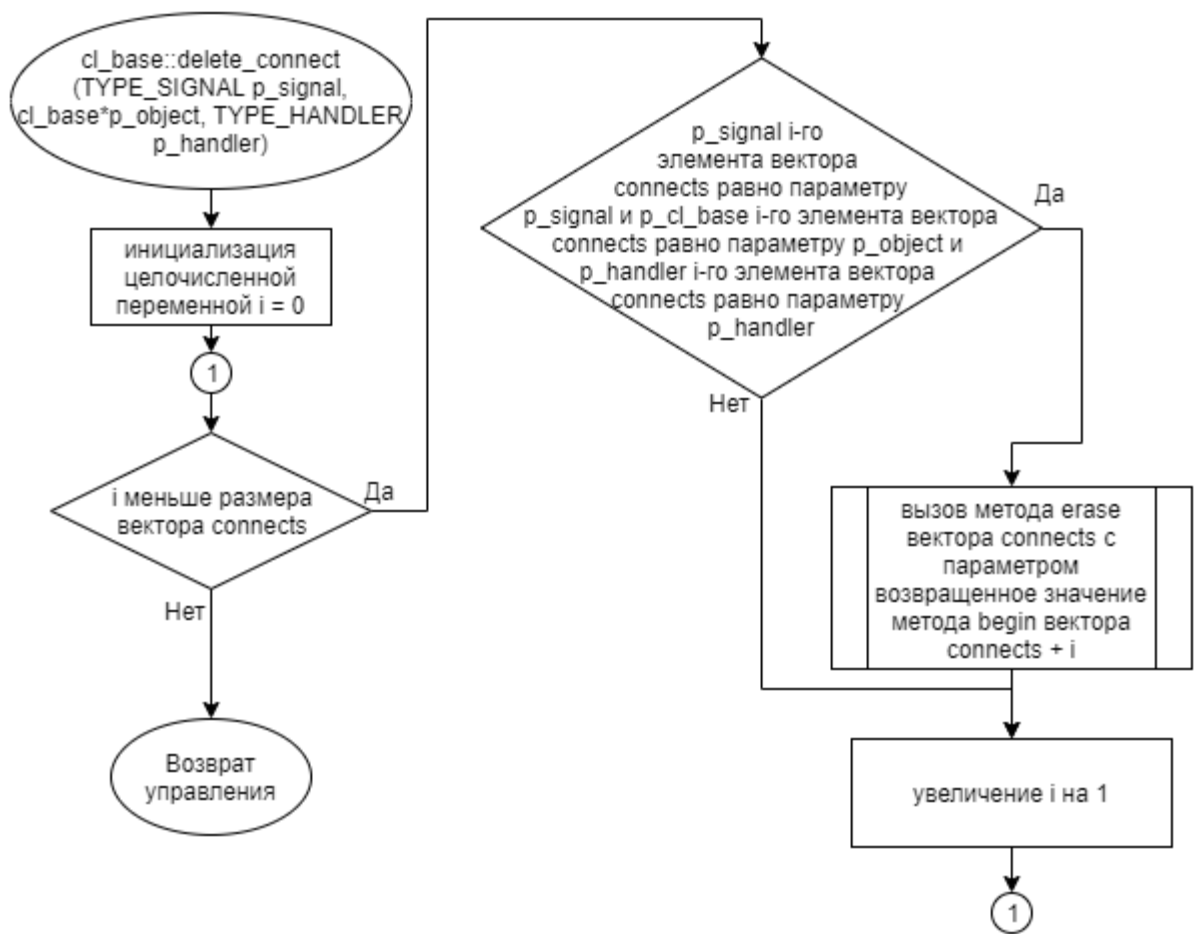


Рисунок 3 – Блок-схема алгоритма

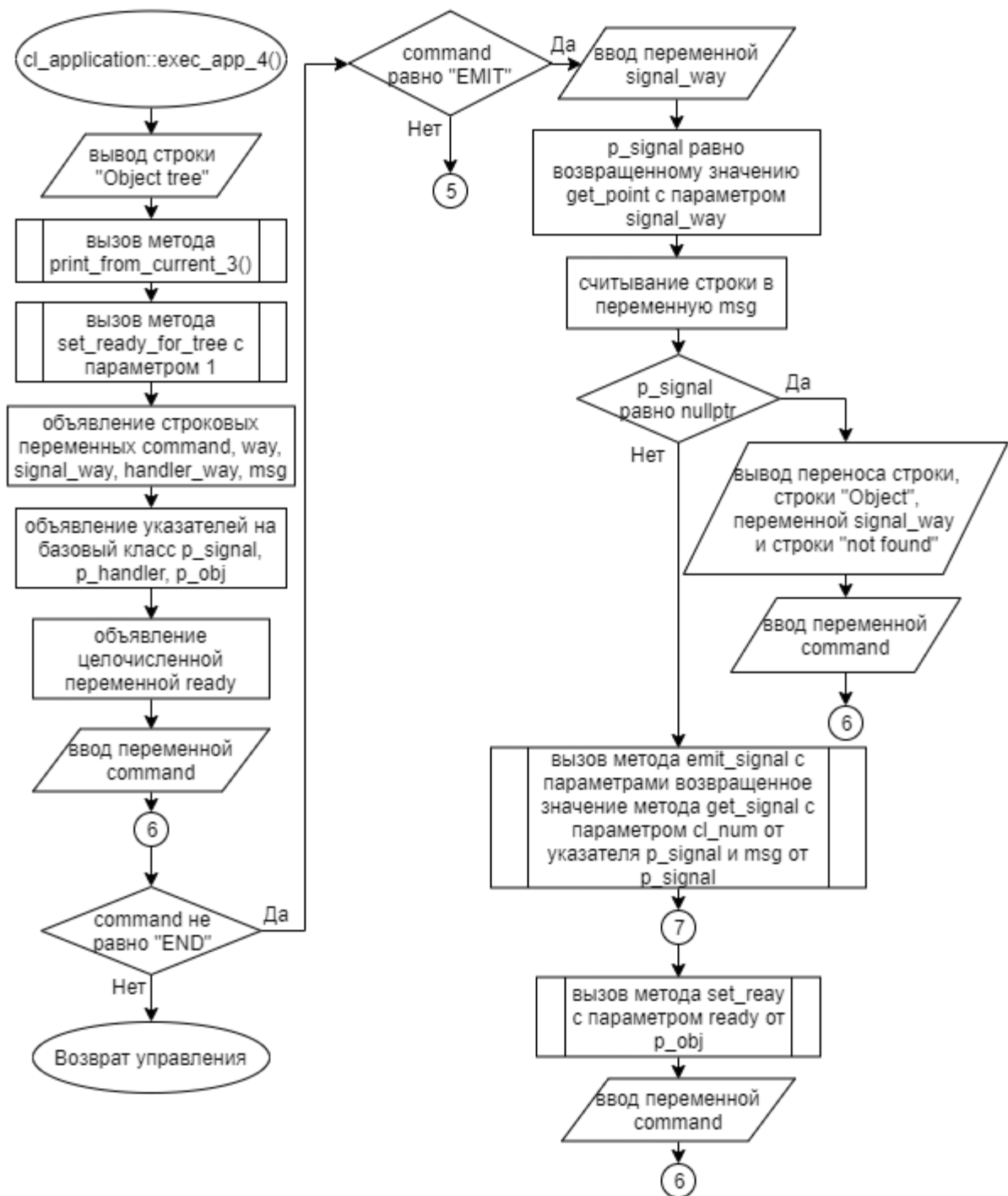


Рисунок 4 – Блок-схема алгоритма

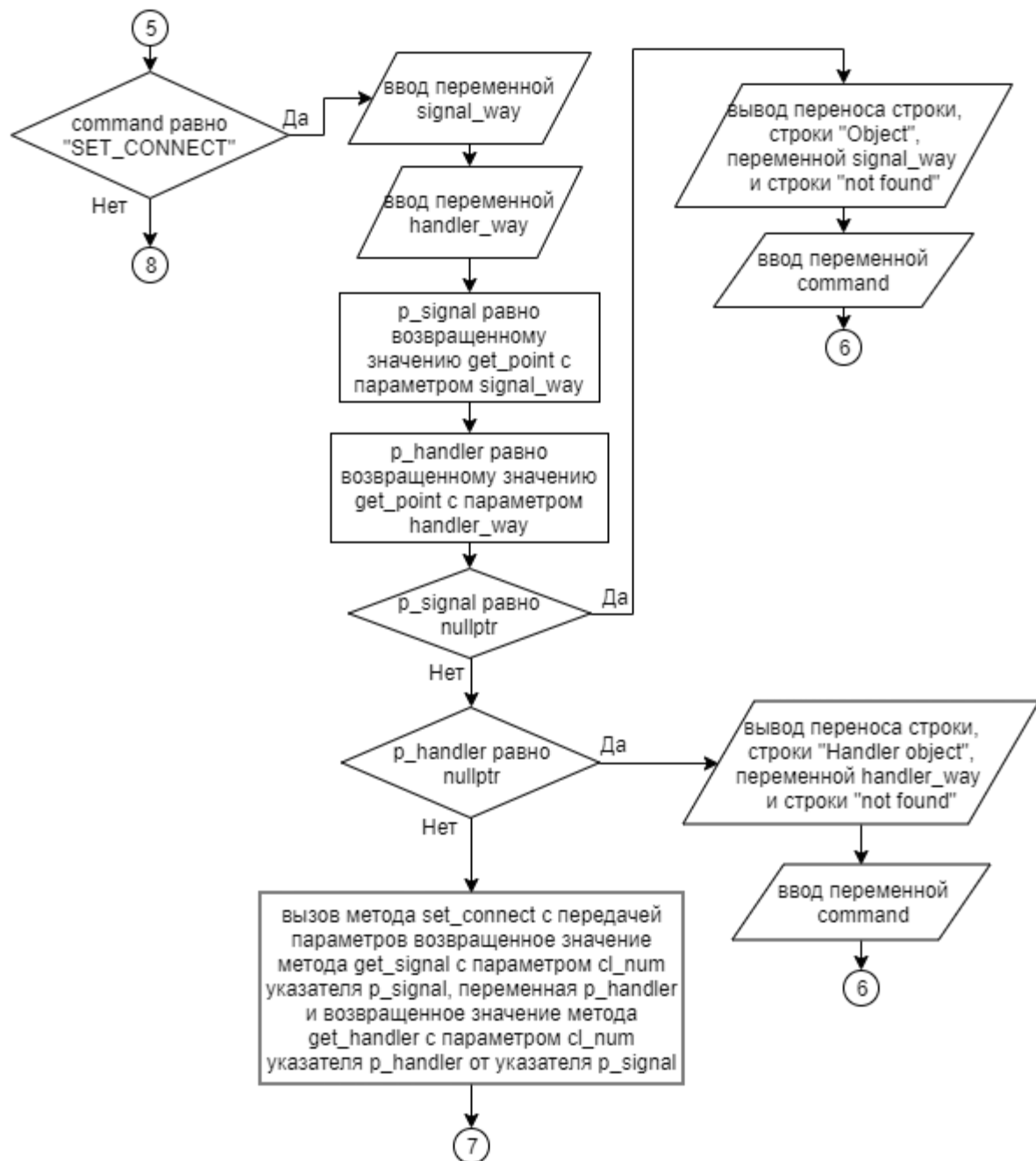


Рисунок 5 – Блок-схема алгоритма

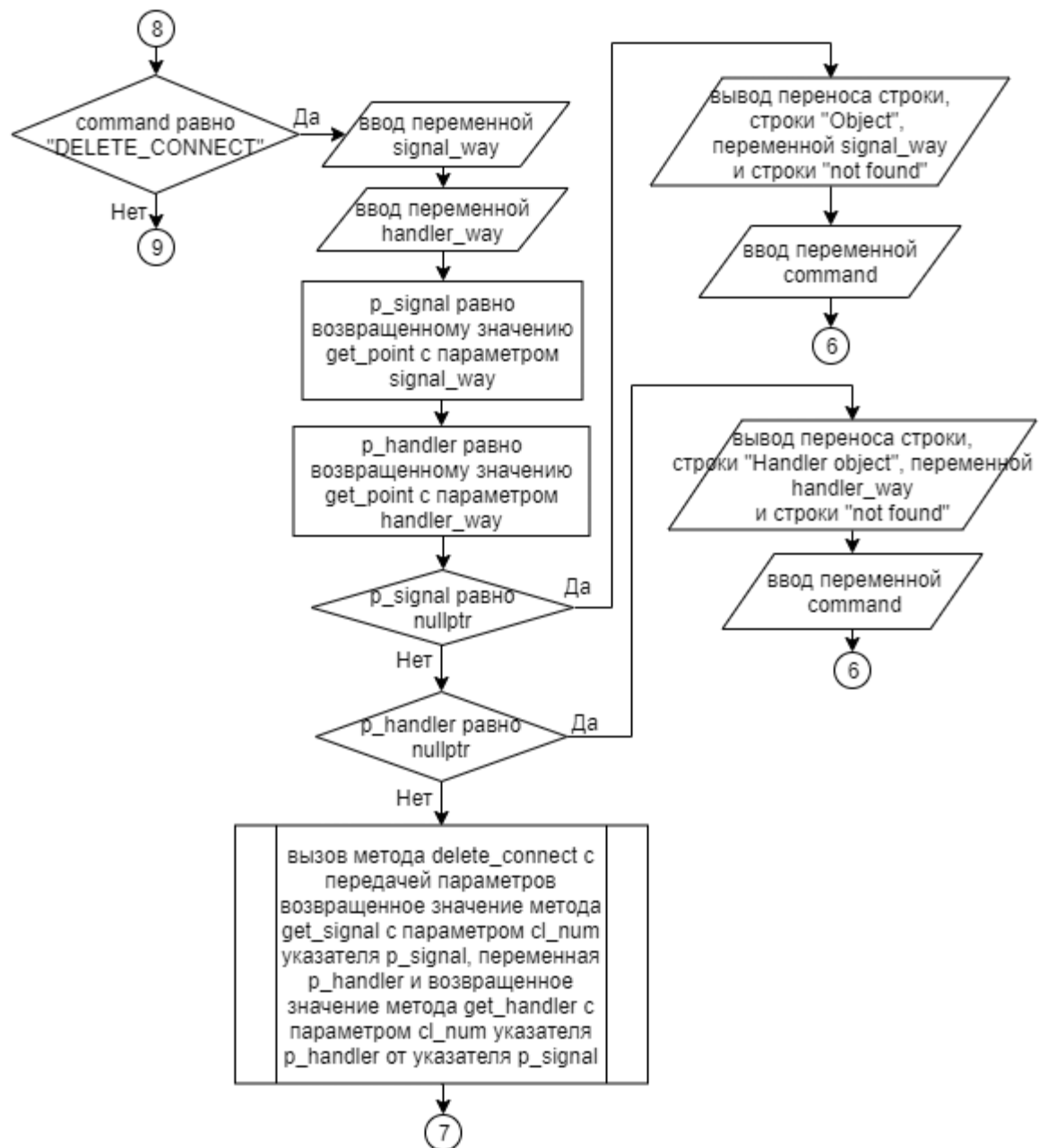


Рисунок 6 – Блок-схема алгоритма

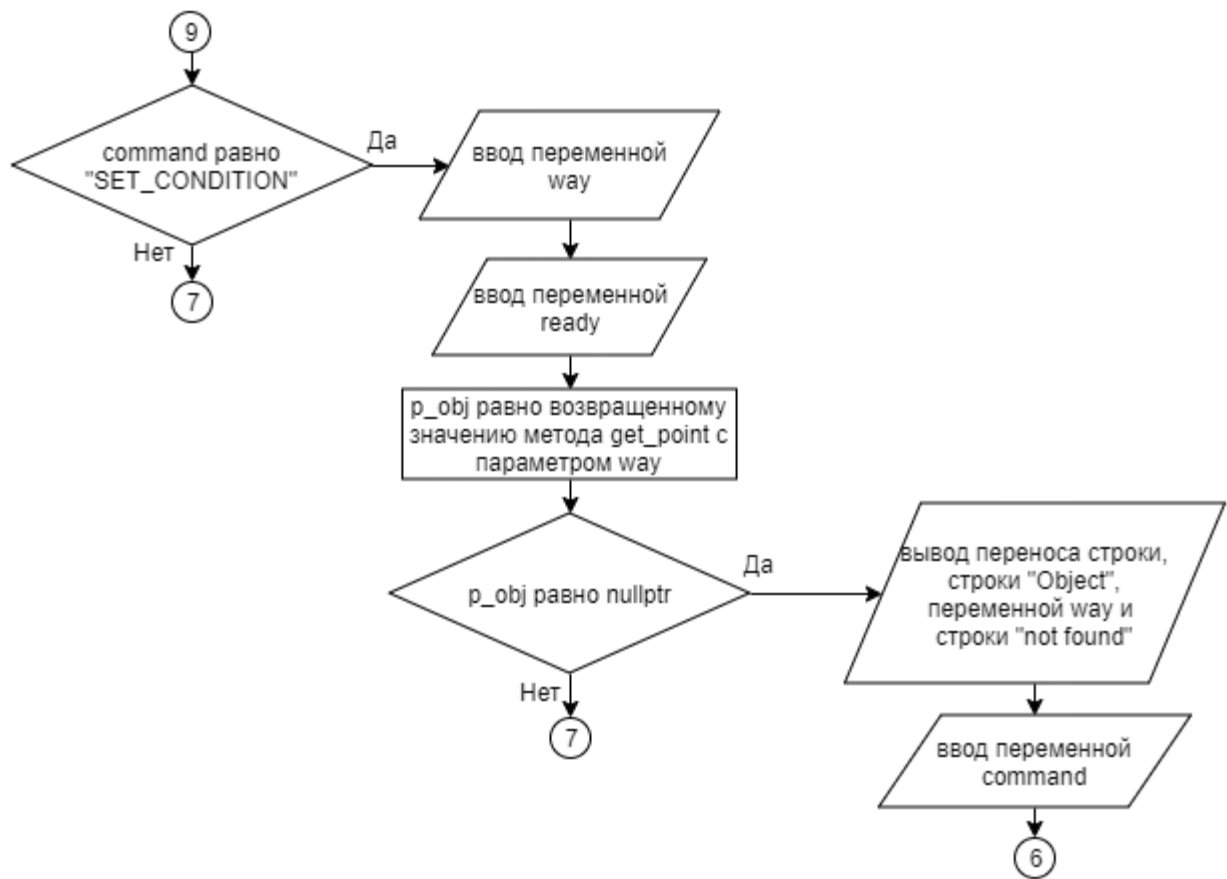


Рисунок 7 – Блок-схема алгоритма



**Рисунок 8 – Блок-схема алгоритма**

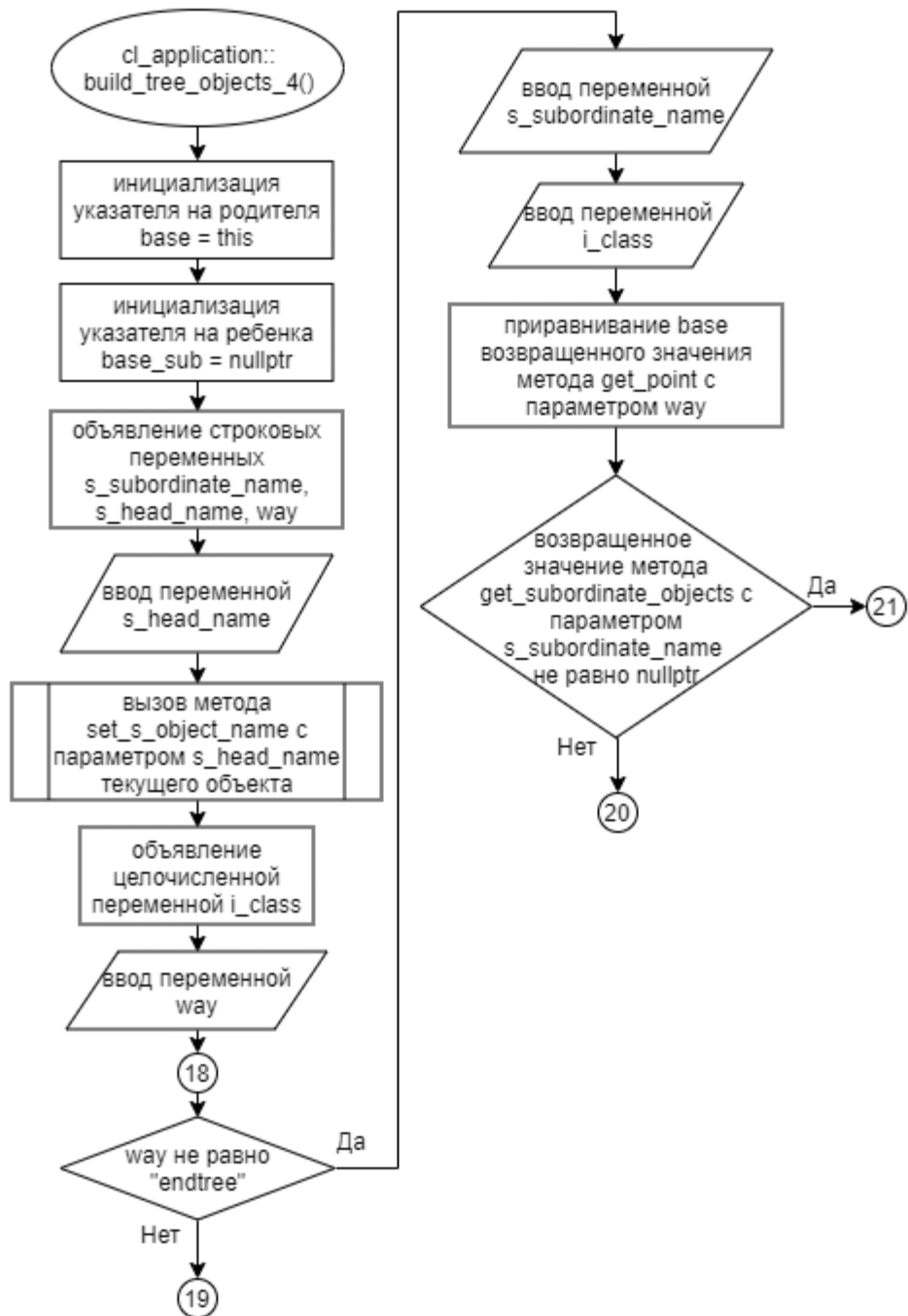


Рисунок 9 – Блок-схема алгоритма

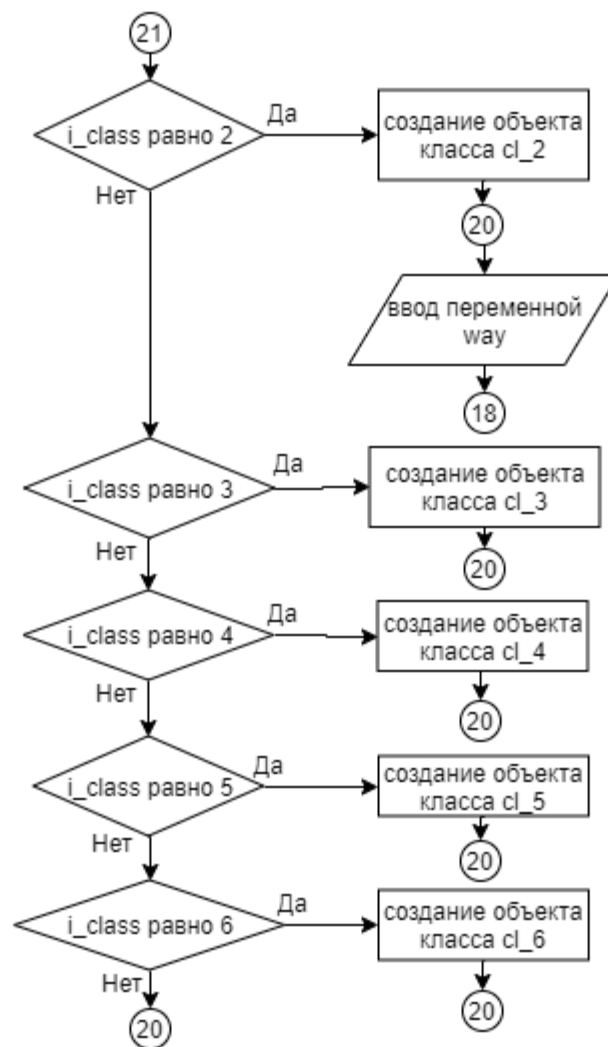


Рисунок 10 – Блок-схема алгоритма



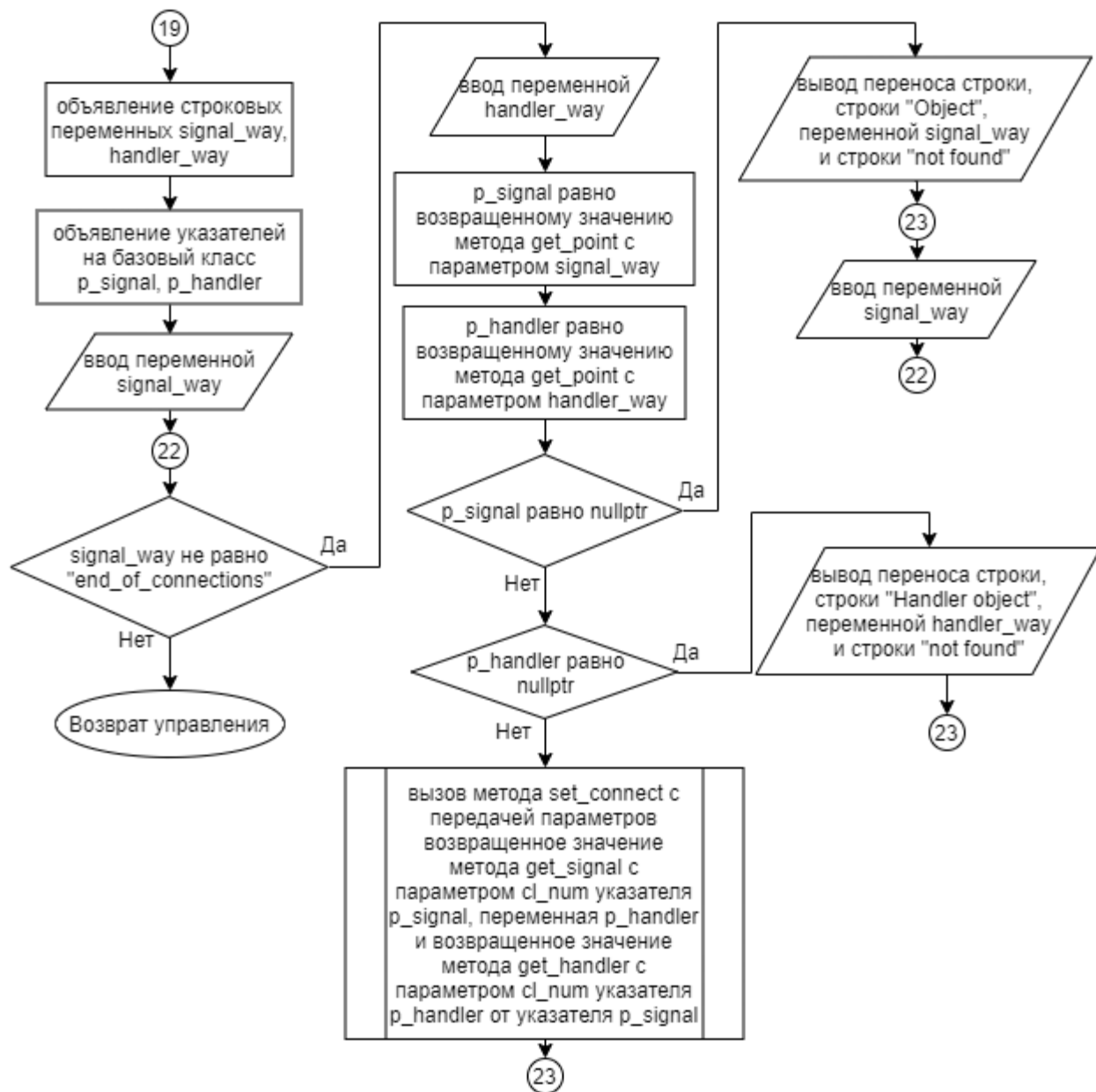


Рисунок 11 – Блок-схема алгоритма

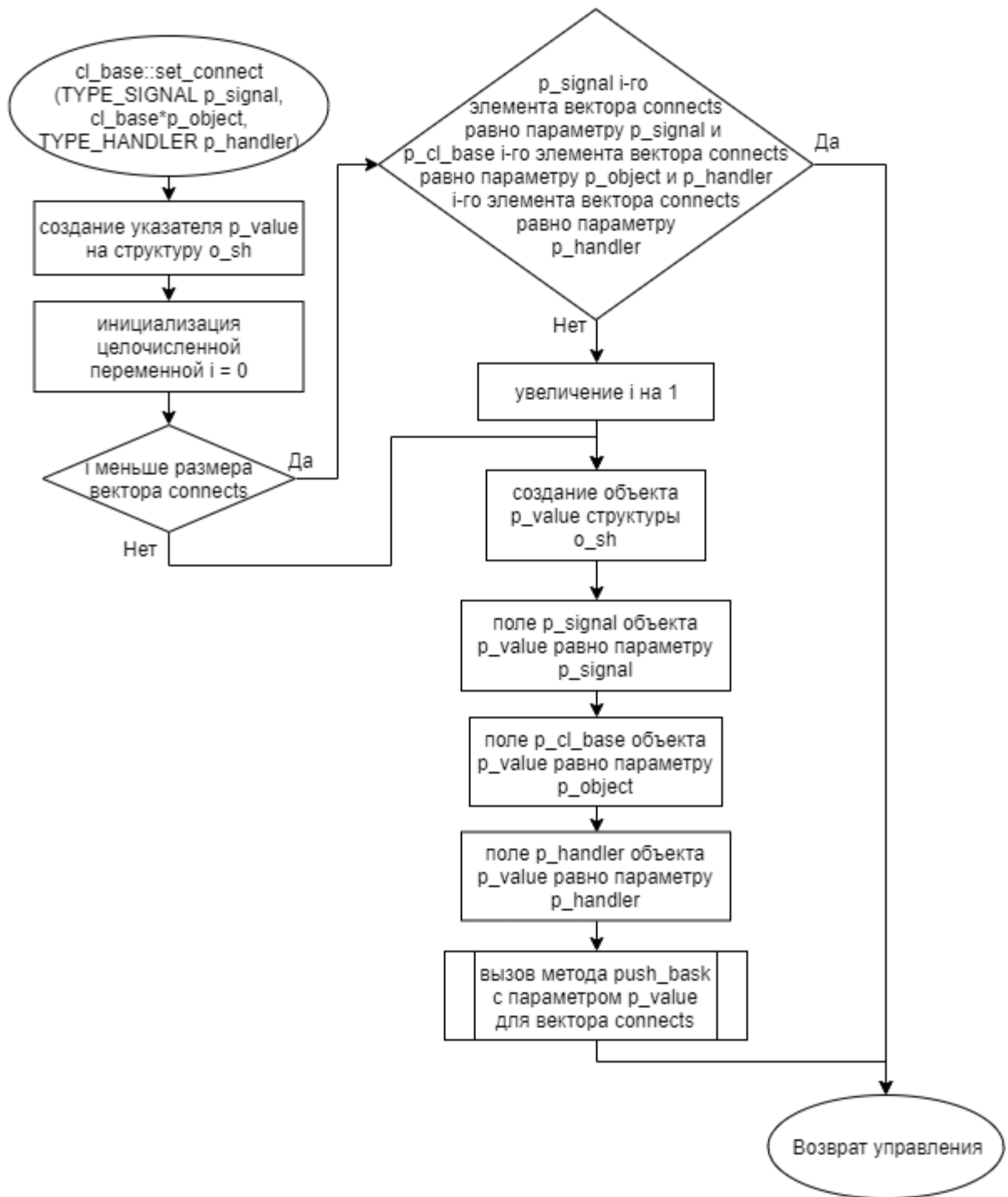


Рисунок 12 – Блок-схема алгоритма

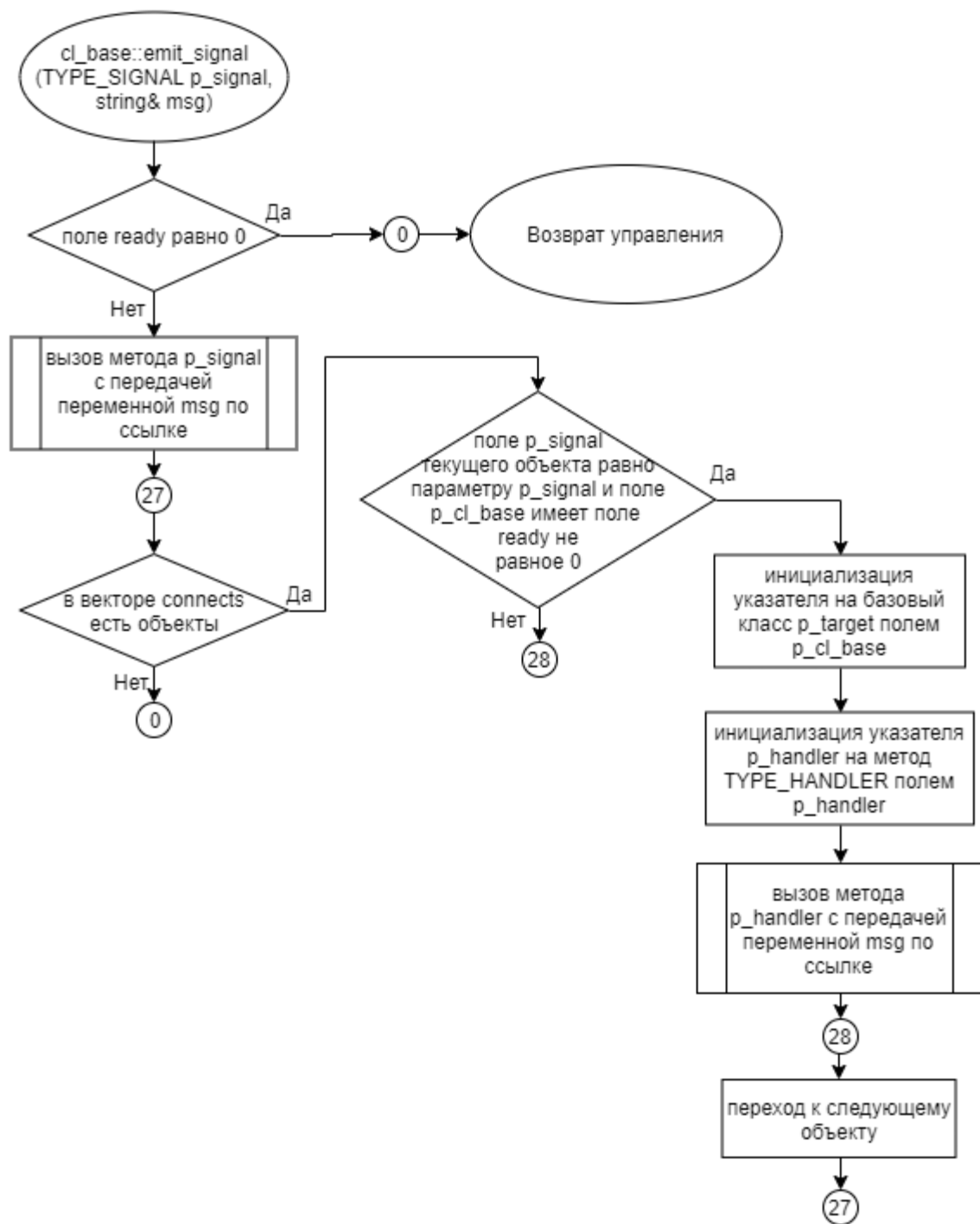


Рисунок 13 – Блок-схема алгоритма

## ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы был разработан и реализован механизм сигналов и обработчиков в базовом классе. Этот механизм предоставляет удобный способ организации взаимодействия объектов, позволяя объектам инициировать сигналы и передавать их нескольким обработчикам. Реализованные методы позволяют установить и удалить связь между сигналом объекта и обработчиком, а также выдать сигнал с передачей данных.

Механизм сигналов и обработчиков имеет широкий спектр применений и может быть использован в различных областях программирования. Он упрощает взаимодействие между объектами, обеспечивая гибкость и модульность. Разработанный механизм предоставляет гибкую архитектуру, полезную при создании сложных систем и приложений.

Применение механизма сигналов и обработчиков позволяет эффективно организовывать взаимодействие объектов, упрощает обмен информацией и повышает переиспользуемость кода. Он предоставляет удобный способ передачи сигналов и данных между объектами, а также гибкое управление взаимодействием в рамках системы.

В итоге, разработанный механизм сигналов и обработчиков является мощным инструментом для организации взаимодействия объектов, способствуя созданию более эффективных, модульных и переиспользуемых систем программного обеспечения.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.
2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: [https://mirea.aco-avrrora.ru/student/files/methodicheskoe\\_posobie\\_dlya\\_laboratornyh\\_rabot\\_3.pdf](https://mirea.aco-avrrora.ru/student/files/methodicheskoe_posobie_dlya_laboratornyh_rabot_3.pdf) (дата обращения 05.05.2021).
3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: [https://mirea.aco-avrrora.ru/student/files/Prilozheniye\\_k\\_methodichke.pdf](https://mirea.aco-avrrora.ru/student/files/Prilozheniye_k_methodichke.pdf) (дата обращения 05.05.2021).
4. Шилдт Г. С++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.
5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

## Приложение 1. Код программы

### Файл cl\_2.cpp

*Листинг 1 – cl\_2.cpp*

```
#include "cl_2.h"
cl_2::cl_2(cl_base * p_head_object, string s_object_name): cl_base(p_head_object,
s_object_name){ //вызов конструктора cl_base с передачей параметров p_head_object,
s_object_name
    cl_num = 2; //поле cl_num равно 2
};
void cl_2::signal_f(string& msg){
    cout << "\nSignal from " << get_way(); //вывод символа переноса строки, строки
"Signal from " и возвращенного значения метода get_way()
    msg += " (class: 2)"; //прибавление строки " (class:2)" к переменной msg
}
void cl_2::handler_f(string& msg){
    cout << "\nSignal to " << get_way() << " Text: " << msg; // вывод символа
переноса строки, строки "Signal to ", возвращенного значения метода get_way(),
строки "Text: " и параметра msg
}
```

### Файл cl\_2.h

*Листинг 2 – cl\_2.h*

```
#ifndef CL_2_H
#define CL_2_H
#include "cl_base.h"

class cl_2 : public cl_base{
public:
    cl_2(cl_base *p_head_object, string s_object_name); //конструктор,
устанавливает значения скрытых свойств
    void signal_f(string& msg); //метод сигнала
    void handler_f(string& msg); //метод обработчика
};
#endif
```

## Файл cl\_3.cpp

Листинг 3 – cl\_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(cl_base * p_head_object, string s_object_name): cl_base(p_head_object,
s_object_name){ //вызов конструктора cl_base с передачей параметров p_head_object,
s_object_name
    cl_num = 3; //поле cl_num равно 3
};
void cl_3::signal_f(string& msg){
    cout << "\nSignal from " << get_way(); //вывод символа переноса строки, строки
"Signal from " и возвращенного значения метода get_way()
    msg += " (class: 3)"; //прибавление строки " (class:3)" к переменной msg
}
void cl_3::handler_f(string& msg){
    cout << "\nSignal to " << get_way() << " Text: " << msg; // вывод символа
переноса строки, строки "Signal to ", возвращенного значения метода get_way(),
строки "Text: " и параметра msg
}
```

## Файл cl\_3.h

Листинг 4 – cl\_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include "cl_base.h"

class cl_3 : public cl_base{
public:
    cl_3(cl_base *p_head_object, string s_object_name); //конструктор,
устанавливает значения скрытых свойств
    void signal_f(string& msg); //метод сигнала
    void handler_f(string& msg); //метод обработчика
};
#endif
```

## Файл cl\_4.cpp

Листинг 5 – cl\_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base * p_head_object, string s_object_name): cl_base(p_head_object,
s_object_name){ //вызов конструктора cl_base с передачей параметров p_head_object,
s_object_name
```

```

        cl_num = 4; //поле cl_num равно 4
    };
void cl_4::signal_f(string& msg){
    cout << "\nSignal from " << get_way(); //вывод символа переноса строки, строки
    "Signal from " и возвращенного значения метода get_way()
    msg += " (class: 4)"; //прибавление строки " (class:4)" к переменной msg
}
void cl_4::handler_f(string& msg){
    cout << "\nSignal to " << get_way() << " Text: " << msg; // вывод символа
переноса строки, строки "Signal to ", возвращенного значения метода get_way(),
строки "Text: " и параметра msg
}

```

## Файл cl\_4.h

Листинг 6 – cl\_4.h

```

#ifndef __CL_4_H
#define __CL_4_H
#include "cl_base.h"

class cl_4 : public cl_base{
public:
    cl_4(cl_base *p_head_object, string s_object_name); //конструктор,
устанавливает значения скрытых свойств
    void signal_f(string& msg); //метод сигнала
    void handler_f(string& msg); //метод обработчика
};
#endif

```

## Файл cl\_5.cpp

Листинг 7 – cl\_5.cpp

```

#include "cl_5.h"
cl_5::cl_5(cl_base * p_head_object, string s_object_name): cl_base(p_head_object,
s_object_name){ //вызов конструктора cl_base с передачей параметров p_head_object,
s_object_name
    cl_num = 5; //поле cl_num равно 5
};
void cl_5::signal_f(string& msg){
    cout << "\nSignal from " << get_way(); //вывод символа переноса строки, строки
    "Signal from " и возвращенного значения метода get_way()
    msg += " (class: 5)"; //прибавление строки " (class:5)" к переменной msg
}
void cl_5::handler_f(string& msg){
    cout << "\nSignal to " << get_way() << " Text: " << msg; // вывод символа

```



```
переноса строки, строки "Signal to ", возвращенного значения метода get_way(), строки "Text: " и параметра msg  
}
```

## Файл cl\_5.h

Листинг 8 – cl\_5.h

```
#ifndef __CL_5_H  
#define __CL_5_H  
#include "cl_base.h"  
  
class cl_5 : public cl_base{  
public:  
    cl_5(cl_base *p_head_object, string s_object_name); //конструктор,  
устанавливает значения скрытых свойств  
    void signal_f(string& msg); //метод сигнала  
    void handler_f(string& msg); //метод обработчика  
};  
#endif
```

## Файл cl\_6.cpp

Листинг 9 – cl\_6.cpp

```
#include "cl_6.h"  
cl_6::cl_6(cl_base * p_head_object, string s_object_name): cl_base(p_head_object,  
s_object_name){ //вызов конструктора cl_base с передачей параметров p_head_object,  
s_object_name  
    cl_num = 6; //поле cl_num равно 6  
};  
void cl_6::signal_f(string& msg){  
    cout << "\nSignal from " << get_way(); //вывод символа переноса строки, строки  
"Signal from " и возвращенного значения метода get_way()  
    msg += " (class: 6)"; //прибавление строки " (class:6)" к переменной msg  
}  
void cl_6::handler_f(string& msg){  
    cout << "\nSignal to " << get_way() << " Text: " << msg; // вывод символа  
переноса строки, строки "Signal to ", возвращенного значения метода get_way(),  
строки "Text: " и параметра msg  
}
```

## Файл cl\_6.h

Листинг 20 – cl\_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"

class cl_6 : public cl_base{
public:
    cl_6(cl_base *p_head_object, string s_object_name); //конструктор,
    устанавливает значения скрытых свойств
    void signal_f(string& msg); //метод сигнала
    void handler_f(string& msg); //метод обработчика
};
#endif
```

## Файл cl\_application.cpp

Листинг 13 – cl\_application.cpp

```
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"

cl_application::cl_application(cl_base* p_head_object):cl_base(p_head_object){
//вызов конструктора cl_base с передачей параметра p_head_object
    cl_num = 1; //поле cl_num равно 1
}
void cl_application::signal_f(string& msg){
    cout << "\nSignal from " << get_way(); //вывод символа переноса строки, строки
    "Signal from " и возвращенного значения метода get_way()
    msg += " (class: 1)"; //прибавление строки " (class:1)" к переменной msg
}
void cl_application::handler_f(string& msg){
    cout << "\nSignal to " << get_way() << " Text: " << msg; // вывод символа
    переноса строки, строки "Signal to ", возвращенного значения метода get_way(),
    строки "Text: " и параметра msg
}
TYPE_SIGNAL cl_application::get_signal(int n){
    switch(n){
        case 1:
            return SIGNAL_D(cl_application::signal_f); //возврат значения
            функции SIGNAL_D с передачей метода signal_f класса cl_application
            break;
        case 2:
            return SIGNAL_D(cl_2::signal_f); //возврат значения функции
```

```

        SIGNAL_D с передачей метода signal_f класса cl_2
            break;
        case 3:
            return SIGNAL_D(cl_3::signal_f); //возврат значения функции
        SIGNAL_D с передачей метода signal_f класса cl_3
            break;
        case 4:
            return SIGNAL_D(cl_4::signal_f); //возврат значения функции
        SIGNAL_D с передачей метода signal_f класса cl_4
            break;
        case 5:
            return SIGNAL_D(cl_5::signal_f); //возврат значения функции
        SIGNAL_D с передачей метода signal_f класса cl_5
            break;
        case 6:
            return SIGNAL_D(cl_6::signal_f); //возврат значения функции
        SIGNAL_D с передачей метода signal_f класса cl_6
            break;
        default:
            return nullptr; //возврат nullptr
    }
}
TYPE_HANDLER cl_application::get_handler(int n){
    switch(n){
        case 1:
            return HANDLER_D(cl_application::handler_f); //возврат значения
            функции HANDLER_D с передачей метода handler_f класса cl_application
            break;
        case 2:
            return HANDLER_D(cl_2::handler_f); //возврат значения функции
        HANDLER_D с передачей метода handler_f класса cl_2
            break;
        case 3:
            return HANDLER_D(cl_3::handler_f); //возврат значения функции
        HANDLER_D с передачей метода handler_f класса cl_3
            break;
        case 4:
            return HANDLER_D(cl_4::handler_f); //возврат значения функции
        HANDLER_D с передачей метода handler_f класса cl_4
            break;
        case 5:
            return HANDLER_D(cl_5::handler_f); //возврат значения функции
        HANDLER_D с передачей метода handler_f класса cl_5
            break;
        case 6:
            return HANDLER_D(cl_6::handler_f); //возврат значения функции
        HANDLER_D с передачей метода handler_f класса cl_6
            break;
        default:
            return nullptr; //возврат nullptr
    }
}
void cl_application::build_tree_objects_4(){
    cl_base * base = this; //инициализация указателя на родителя base = this
    cl_base * base_sub = nullptr; //инициализация указателя на ребенка base_sub =
    nullptr
    string s_subordinate_name, s_head_name, way; //объявление переменных

```

```

строкового типа s_subordinate_name, s_head_name и way
    cin >> s_head_name; //ввод переменной s_head_name
    this -> set_s_object_name(s_head_name); //вызов метода
set_object_name(s_head_name)
    int i_class; //объявление целочисленной переменной i_class
    cin >> way; //ввод переменной way
    while (!(way == "endtree")){ //way не равно "endtree"
        cin >> s_subordinate_name; //ввод переменной s_subordinate_name
        cin >> i_class; //ввод переменной i_class
        base = get_point(way); //присваивание base возвращенного значения метода
get_point(way)
        if (!base -> get_subordinate_objects(s_subordinate_name)){
//возвращенное значение метода get_subordinate_objects с параметром
s_subordinate_name не равно nullptr
            switch(i_class){
                case 2:
                    base_sub = new cl_2(base, s_subordinate_name);
//создание объекта класса cl_2
                    break;
                case 3:
                    base_sub = new cl_3(base, s_subordinate_name);
//создание объекта класса cl_3
                    break;
                case 4:
                    base_sub = new cl_4(base, s_subordinate_name);
//создание объекта класса cl_4
                    break;
                case 5:
                    base_sub = new cl_5(base, s_subordinate_name);
//создание объекта класса cl_5
                    break;
                case 6:
                    base_sub = new cl_6(base, s_subordinate_name);
//создание объекта класса cl_6
                    break;
            }
        }
        cin >> way; //ввод переменной way
    }
    string signal_way, handler_way; //объявление строковых переменных signal_way,
handler_way
    cl_base *p_signal, *p_handler; //объявление указателей на базовый класс
p_signal, p_handler
    cin >> signal_way; //ввод переменной signal_way
    while (!(signal_way == "end_of_connections")){ //signal_way не равно
"end_of_connections"
        cin >> handler_way; //ввод переменной handler_way
        p_signal = get_point(signal_way); //p_signal равно возвращенному
значению метода get_point с параметром signal_way
        p_handler = get_point(handler_way); //p_handler равно возвращенному
значению метода get_point с параметром handler_way
        if (!p_signal){ //p_signal равно nullptr
            cout << "\nObject " << signal_way << " not found"; //вывод переноса
строки, строки "Object", переменной signal_way и строки "not found"
            continue;
        }
        if (!p_handler){ // p_handler равно nullptr

```

```

        cout << "\nHandler object " << handler_way << " not found"; //вывод
переноса строки, строки "Handler object", переменной handler_way и строки "not
found"

        continue;
    }
    p_signal -> set_connect(get_signal(p_signal -> cl_num), p_handler,
get_handler(p_handler -> cl_num)); //вызов метода set_connect с передачей параметров
возвращенное значение метода get_signal с параметром cl_num указателя p_signal,
переменная p_handler и возвращенное значение метода get_handler с параметром cl_num
указателя p_handler от указателя p_signal
    cin >> signal_way; //ввод переменной signal_way
}
}
int cl_application::exec_app_4(){
    cout << "Object tree"; //вывод строки "Object tree"
    print_from_current_3(); //вызов метода print_from_current_3()
    set_ready_for_tree(1); //вызов метода set_ready_for_tree с параметром 1
    string command, way, signal_way, handler_way, msg; //объявление строковых
переменных command, way, signal_way, handler_way, msg
    cl_base *p_signal, *p_handler, *p_obj; //объявление указателей на базовый
класс p_signal, p_handler, p_obj
    int ready; //объявление целочисленной переменной ready
    cin >> command; //ввод переменной command
    while (command != "END"){ //command не равно "END"
        if (command == "EMIT"){ //command равно "EMIT"
            cin >> signal_way; //ввод переменной signal_way
            p_signal = get_point(signal_way); //p_signal равно возвращенному
значению get_point с параметром signal_way
            getline(cin, msg); //считывание строки в переменную msg
            if (!p_signal){ //p_signal равно nullptr
                cout << "\nObject " << signal_way << " not found"; //вывод
переноса строки, строки "Object", переменной signal_way и строки "not found"
                cin >> command; //ввод переменной command
                continue;
            }
            p_signal -> emit_signal(get_signal(p_signal -> cl_num), msg);
//вызов метода emit_signal с параметрами возвращенное значение метода get_signal с
параметром cl_num от указателя p_signal и msg от p_signal
        } else if (command == "SET_CONNECT"){ //command равно "SET_CONNECT"
            cin >> signal_way; //ввод переменной signal_way
            cin >> handler_way; //ввод переменной handler_way
            p_signal = get_point(signal_way); //p_signal равно возвращенному
значению get_point с параметром signal_way
            p_handler = get_point(handler_way); //p_handler равно
возвращенному значению get_point с параметром handler_way
            if (!p_signal){ //p_signal равно nullptr
                cout << "\nObject " << signal_way << " not found"; //вывод
переноса строки, строки "Object", переменной signal_way и строки "not found"
                cin >> command; //ввод переменной command
                continue;
            }
            if (!p_handler){ //p_handler равно nullptr
                cout << "\nHandler object " << handler_way << " not found";
//вывод переноса строки, строки "Handler object", переменной handler_way и строки
"not found"

                cin >> command; //ввод переменной command
                continue;
            }
        }
    }
}

```

```

    }
    p_signal -> set_connect(get_signal(p_signal -> cl_num),
p_handler, get_handler(p_handler -> cl_num)); //вызов метода set_connect с передачей
параметров возвращенное значение метода get_signal с параметром cl_num указателя
p_signal, переменная p_handler и возвращенное значение метода get_handler с
параметром cl_num указателя p_handler от указателя p_signal
    } else if (command == "DELETE_CONNECT"){ //command равно
"DELETE_CONNECT"
        cin >> signal_way; //ввод переменной signal_way
        cin >> handler_way; //ввод переменной handler_way
        p_signal = get_point(signal_way); //p_signal равно возвращенному
значению get_point с параметром signal_way
        p_handler = get_point(handler_way); //p_handler равно
возвращенному значению get_point с параметром handler_way
        if (!p_signal){ //p_signal равно nullptr
            cout << "\nObject " << signal_way << " not found"; //вывод
переноса строки, строки "Object", переменной signal_way и строки "not found"
            cin >> command; //ввод переменной command
            continue;
        }
        if (!p_handler){ //p_handler равно nullptr
            cout << "\nHandler object " << handler_way << " not found";
//вывод переноса строки, строки "Handler object", переменной handler_way и строки
"not found"
            cin >> command; //ввод переменной command
            continue;
        }
        p_signal -> delete_connect(get_signal(p_signal -> cl_num),
p_handler, get_handler(p_handler -> cl_num)); //вызов метода delete_connect с
передачей параметров возвращенное значение метода get_signal с параметром cl_num
указателя p_signal, переменная p_handler и возвращенное значение метода get_handler
с параметром cl_num указателя p_handler от указателя p_signal
    } else if (command == "SET_CONDITION"){ //command равно "SET_CONDITION"
        cin >> way; //ввод переменной way
        cin >> ready; //ввод переменной ready
        p_obj = get_point(way); //p_obj равно возвращенному значению
метода get_point с параметром way
        if (!p_obj){ //p_obj равно nullptr
            cout << "\nObject " << way << " not found"; //вывод переноса
строки, строки "Object", переменной way и строки "not found"
            cin >> command; //ввод переменной command
            continue;
        }
        p_obj -> set_ready(ready); //вызов метода set_reay с параметром
ready от p_obj
    }
    cin >> command; //ввод переменной command
}
return 0;
}

```

## Файл cl\_application.h

Листинг 42 – cl\_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H
#include "cl_base.h"

class cl_application : public cl_base{
public:
    cl_application(cl_base* p_head_object); //конструктор, устанавливает значения
скрытых свойств
    void signal_f(string& msg); //метод сигнала
    void handler_f(string& msg); //метод обработчика
    TYPE_SIGNAL get_signal(int n); //выдача сигнала по номеру класса
    TYPE_HANDLER get_handler(int n); //выдача метода обработчика по номеру класса
    void build_tree_objects_4(); //метод построения исходного дерева иерархии
объектов (конструирование моделируемой системы)
    int exec_app_4(); //метод запуска приложения (начало функционирования системы,
выполнение алгоритма решения задачи)
};
#endif
```

## Файл cl\_base.cpp

Листинг 53 – cl\_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head_object, string s_object_name){
    this -> p_head_object = p_head_object; //присвоение полю p_head_object
значения параметра p_head_object
    this -> s_object_name = s_object_name; //присвоение полю s_object_name
значения параметра s_object_name
    if (get_p_head_object()){ //возвращенное значение метода get_p_head_object не
равно 0
        p_head_object -> subordinate_objects.push_back(this); //добавление
адреса текущего объекта с помощью метода push_back в вектор subordinate_objects
объекта по указателю p_head_object
    }
}

string cl_base::get_s_object_name(){
    return s_object_name; //возврат поля s_object_name
}

cl_base * cl_base::get_p_head_object(){
    return p_head_object; //возврат поля p_head_object
}

cl_base * cl_base::get_subordinate_objects(string s_name){
    for (int i = 0; i < subordinate_objects.size(); i++){ //i меньше размера
вектора subordinate_objects
```

```

        if (subordinate_objects[i] -> get_s_object_name() == s_name){
//возвращенное значение метода get_s_object_name от i-го элемента вектора
subordinate_objects равно значению параметра s_name
        return subordinate_objects[i]; //возврат i-го элемента вектора
subordinate_objects
        }
    }
    return nullptr; //возврат nullptr
}
bool cl_base::set_s_object_name(string s_new_name){
    if (get_p_head_object()){ //возвращаемое значение метода get_p_head_object не
равно 0
        for (int i = 0; i < get_p_head_object() -> subordinate_objects.size();
i++){ //i меньше размера вектора subordinate_objects
            if (get_p_head_object() -> subordinate_objects[i] ->
get_s_object_name() == s_new_name){return false;} //возвращенное значение метода
get_p_head_object от i-го элемента вектора subordinate_objects равно значению
параметра s_new_name
            }
        }
        this -> s_object_name = s_new_name; //поле s_object_name равно значению
параметра s_new_name
        return true;
    }
}
cl_base::~cl_base(){
    for (int i = 0; i < subordinate_objects.size(); i++){delete
subordinate_objects[i];} // i меньше размера вектора subordinate_objects,
удаление i-го элемента вектора subordinate_objects
}
int cl_base::count(string s_name){
    int cnt = 0; //инициализация целочисленной переменной cnt = 0
    if (this -> get_s_object_name() == s_name){cnt++;} //возвращенное значение
метода get_s_object_name от текущего объекта равно параметру s_name, увеличение cnt
на 1
    for (auto p_sub_object : subordinate_objects){ //в векторе subordinate_objects
есть объекты
        cnt += p_sub_object -> count(s_name); //прибавление к переменной cnt
возвращенного значения метода count с параметром s_name от p_sub_object
    }
    return cnt; //возврат значения переменной cnt
}
cl_base* cl_base::search_object(string s_name){
    if (this -> get_s_object_name() == s_name){return this;} //возвращенное
значение метода get_s_object_name от текущего объекта равно параметру s_name
    for (auto p_sub_object : subordinate_objects){ //в векторе subordinate_objects
есть объекты
        cl_base* p_found = p_sub_object -> search_object(s_name); //присвоение
указателю p_found возвращенного значения метода search_object с параметром s_name
от p_sub_object
        if (p_found != nullptr){return p_found;} //указатель p_found не равен
nullptr, возврат указателя p_found
    }
    return nullptr; //возврат nullptr
}
cl_base* cl_base::find_object_from_current(string s_name){
    if (count(s_name) != 1){return nullptr;} //возвращенное значение метода count
с параметром s_name не равно 1

```



```

        return search_object(s_name); //возврат возвращенного значения метода
search_object с параметром s_name
    }
    cl_base* cl_base::find_root_from_current(){
        if (this -> get_p_head_object() == nullptr){return this;} //возвращенное
значение метода get_p_head_object от текущего объекта равно nullptr
        return get_p_head_object() -> find_root_from_current(); //возврат
возвращенного значения метода find_root_from_current от возвращенного значения
метода get_p_head_object
    }
    cl_base* cl_base::find_object_from_root(string s_name){
        return find_root_from_current() -> find_object_from_current(s_name);
//возврат возвращенного значения метода find_object_from_current с параметром s_name
от возвращенного значения метода find_root_from_current
    }
    int cl_base::get_ready(){
        return this -> ready; //возврат поля ready от текущего объекта
    }
    void cl_base::set_ready(int i_ready){
        if (i_ready != 0){ //параметра i_ready не равен 0
            cl_base* p = this -> get_p_head_object(); //инициализация указателя p
возвращенным значением метода get_p_head_object от текущего объекта
            while (p){ //указатель p не равен nullptr
                if (p -> get_ready() == 0){return;} //возвращенное значение метода
get_ready от указателя p равно 0
                p = p -> get_p_head_object(); //присвоение указателю p
возвращенного значения метода get_p_head_object от указателя p
            }
            this -> ready = i_ready; //поле ready равно значению параметра i_ready
        } else {
            this -> ready = i_ready; //поле ready равно значению параметра i_ready
            for (auto p_sub_object : subordinate_objects){p_sub_object ->
set_ready(i_ready);} //вызов метода set_ready с параметром i_ready от p_sub_object
        }
    }
    cl_base* cl_base::get_point(string way){
        if (way == ""){return nullptr;} //way равно "", возврат nullptr
        else if (way == "/"){return find_root_from_current();} //way равно "/",
возврат возвращенного значения метода find_root_from_current()
        else if (way == "."){return this;} //way равно ".", возврат указателя this
        else if ((way[0] == '/') && (way[1] == '/')){return
find_object_from_root(way.substr(2));} //0-ой символ переменной way равен '/' и 1-
ый символ переменной way равен '/', возврат возвращенного значения метода
find_object_from_root с параметром возвращенное подстроки way со 2 элемента с
помощью метода substr
        else if (way[0] == '.'){return find_object_from_current(way.substr(1));} //0-
ой символ переменной way равен '.', возврат возвращенного значения метода
find_object_from_root с параметром возвращенной подстроки way с 1 элемента с помощью
метода substr
        else if (way[0] == '/'){ //0-ой символ переменной way равен '/'
            cl_base* p = find_root_from_current(); //инициализация указателя на
базовый класс p возвращенным значением метода find_root_from_current()
            string name = ""; //инициализация строковой переменной name = ""
            way = way.substr(1); //присвоение переменной way возвращенного значения
метода substr(1) для way
            for (char ch : way){ // в строке way есть символы ch
                if (ch == '/'){ //ch равно '/'

```

```

        p = p -> get_subordinate_objects(name); //присвоение
указателю p возвращенного значения метода get_subordinate_objects с параметром name
от указателя p
        if (!p) {return p;} //p равно nullptr, возврат p
        name = ""; //присвоение переменной name ""
    }else{name += ch;} //присвоение переменной name значения name +
ch
    }
    p = p -> get_subordinate_objects(name); //присвоение указателю p
возвращенного значения метода get_subordinate_objects с параметром name от указателя
p
    return p; //возврат p
}else if(way[0] != '/') { //0-ой символ переменной way не равен '/'
    cl_base* p = this; //инициализация указателя на базовый класс p = this
    string name = ""; //инициализация строковой переменной name = ""
    for (char ch : way){ // в строке way есть символы ch
        if (ch == '/') { //ch равно '/'
            p = p -> get_subordinate_objects(name); //присвоение
указателю p возвращенного значения метода get_subordinate_objects(name)
            if (!p) {return p;} //p равно nullptr, возврат p
            name = ""; //присвоение переменной name ""
        }else{name += ch;} //присвоение переменной name значения name +
ch
        }
        p = p -> get_subordinate_objects(name); //присвоение указателю p
возвращенного значения метода get_subordinate_objects с параметром name от указателя
p
        return p; //возврат p
    }else{return nullptr;} //возврат nullptr
}
void cl_base::print_from_current_3(){
    int tab = 0; //инициализация целочисленной переменной tab = 0
    cl_base* p = this -> get_p_head_object(); //инициализация указателя p на
базовый класс cl_base возвращенным значением метода get_p_head_object от текущего
объекта
    while (p){ //p не равно nullptr
        p = p -> get_p_head_object(); //присвоение указателю p возвращенного
значения метода get_p_head_object от указателя p
        tab++; //увеличение tab на 1
    }
    cout << endl << string(tab*4, ' ') << this -> get_s_object_name(); //вывод
переноса строки, строки tab*4, ' ' и возвращенного значения метода get_s_object_name
от текущего объекта
    for (auto p_sub_object : subordinate_objects){p_sub_object ->
print_from_current_3();} //в векторе subordinate_objects есть объекты, вызов метода
print_from_current_3 от p_sub_object
}
void cl_base::set_connect(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_handler){
    o_sh* p_value; //создание указателя p_value на структуру o_sh
    for (int i = 0; i < connects.size(); i++){
        if (connects[i] -> p_signal == p_signal && connects[i] -> p_cl_base ==
p_object && connects[i] -> p_handler == p_handler){return;} //p_signal i-го элемента
вектора connects равно параметру p_signal и p_cl_base i-го элемента вектора connects
равно параметру p_object и p_handler i-го элемента вектора connects равно параметру
p_handler
    }
}

```

```

        p_value = new o_sh(); //создание объекта p_value структуры o_sh
        p_value -> p_signal = p_signal; //поле p_signal объекта p_value равно
параметру p_signal
        p_value -> p_cl_base = p_object; //поле p_cl_base объекта p_value равно
параметру p_object
        p_value -> p_handler = p_handler; //поле p_handler объекта p_value равно
параметру p_handler
        connects.push_back(p_value); //вызов метода push_back с параметром p_value
для вектора connects
    }
void cl_base::delete_connect(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_handler){
    for (int i = 0; i < connects.size(); i++){
        if (connects[i] -> p_signal == p_signal && connects[i] -> p_cl_base ==
p_object && connects[i] -> p_handler == p_handler){ //p_signal i-го элемента вектора
connects равно параметру p_signal и p_cl_base i-го элемента вектора connects равно
параметру p_object и p_handler i-го элемента вектора connects равно параметру
p_handler
            connects.erase(connects.begin() + i); //вызов метода erase
вектора connects с параметром возвращенное значение метода begin вектора connects +
i
        }
    }
}
void cl_base::emit_signal(TYPE_SIGNAL p_signal, string& msg){
    if (!ready){return;} //поле ready равно 0
    (this ->* p_signal)(msg); //вызов метода p_signal с передачей переменной msg
по ссылке
    for (auto connection : connects){ //в векторе connects есть объекты
        if (connection -> p_signal == p_signal && connection -> p_cl_base ->
ready){ //поле p_signal текущего объекта равно параметру p_signal и поле p_cl_base
имеет поле ready не равное 0
            cl_base* p_target = connection -> p_cl_base; //инициализация
указателя на базовый класс p_target полем p_cl_base
            TYPE_HANDLER p_handler = connection -> p_handler; //инициализация
указателя p_handler на метод TYPE_HANDLER полем p_handler
            (p_target ->* p_handler)(msg); //вызов метода p_handler с
передачей переменной msg по ссылке
        }
    }
}
string cl_base::get_way(){
    string way = "/"; //инициализация строковой переменной way, равной "/"
    cl_base* p = this; //инициализация указателя на базовый класс p равно this
    if (!p -> get_p_head_object()){return way;} //возвращенное значение метода
get_p_head_object() равно nullptr, возврат переменной way
    way += p -> get_s_object_name(); //добавление к переменной way возвращенного
значения метода get_s_object_name()
    while (p -> get_p_head_object() -> get_p_head_object()){ //возвращенное
значение метода get_p_head_object() от возвращенного значения метода
get_p_head_object() равно nullptr
        p = p -> get_p_head_object(); //указатель p равен возвращенному значению
метода get_p_head_object() от указателя p
        way = "/" + p -> get_s_object_name() + way; //переменная way равно
строке "/", возвращенному значению метода get_s_object_name() и переменной way
    }
    return way; //возврат переменной way
}

```

```

}
void cl_base::set_ready_for_tree(int ready){
    this -> ready = ready; //поле ready текущего объекта равно параметру ready
    for (auto p_sub : subordinate_objects){p_sub -> set_ready_for_tree(ready);}
//вызов метода set_ready_for_tree с параметром ready для текущего объекта
}

```

## Файл cl\_base.h

*Листинг 64 – cl\_base.h*

```

#ifndef CL_BASE_H
#define CL_BASE_H
#include <string>
#include <vector>
#include <iostream>
using namespace std;

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER) (&handler_f)

class cl_base;

typedef void(cl_base::*TYPE_SIGNAL) (string&);
typedef void(cl_base::*TYPE_HANDLER) (string&);

struct o_sh{
    TYPE_SIGNAL p_signal; //указатель на метод сигнала
    cl_base* p_cl_base; //указатель на целевой объект
    TYPE_HANDLER p_handler; //указатель на метод обработчика
};

class cl_base{
protected:
    string s_object_name; //наименование объекта
    cl_base * p_head_object; //указатель на головной объект для текущего объекта
    vector < cl_base * > subordinate_objects; //массив указателей на объекты,
    подчиненные к текущему объекту в дереве иерархии
    int ready; //состояние готовности объекта
    vector <o_sh*> connects; //вектор хранения установленных связей
public:
    cl_base(cl_base* p_head_object, string s_object_name = "Base_object");
//параметризованный конструктор
    bool set_s_object_name(string s_new_name); //метод редактирования имени
    объекта
    string get_s_object_name(); //метод получения имени объекта
    cl_base * get_p_head_object(); //метод получения указателя на головной объект
    текущего объекта
    cl_base * get_subordinate_objects(string s_object_name); //метод получения
    указателя на подчиненный объект по его имени
    ~cl_base(); //для удаления дерева целиком (вызывает деструктор для всех
    объектов, подчиненных текущему)
    cl_base* find object from root(string s_name); //поиск объекта по имени от

```

```

корневого объекта
    cl_base*search_object(string s_name); //метод поиска объекта по имени от
текущего объекта
    int count(string s_name); //метод определения количества вхождений объектов с
данным именем от текущего объекта
    cl_base*find_object_from_current(string s_name); //метод поиска объекта по
имени от текущего объекта
    void set_ready(int i_ready); //установка готовности объекта
    int get_ready(); //получение состояние готовности объекта
    cl_base* find_root_from_current(); //метод поиска корневого объекта
    cl_base* get_point(string way); //метод получения указателя на любой объект в
составе дерева иерархии объектов согласно пути (координаты)
    void print_from_current_3(); //метод вывода иерархии объектов (дерева или
ветки) от текущего объекта
    int cl_num; //номер класса объекта
    void set_connect(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_handler); //метод установления связи между сигналом текущего объекта и
обработчиком целевого объекта
    void delete_connect(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_handler); //метод удаления связи между сигналом текущего объекта и обработчиком
целевого объекта
    void emit_signal(TYPE_SIGNAL p_signal, string& msg); //метод выдачи сигнала
от текущего объекта с передачей строковой переменной
    string get_way(); //метод получение абсолютного пути до текущего объекта
    void set_ready_for_tree(int ready); //метод установления состояния готовности
для всех объектов
};
#endif

```

## Файл main.cpp

*Листинг 75 – main.cpp*

```

#include "cl_application.h"
int main(){
    cl_application      ob_cl_application(nullptr);           //создание      объекта
ob_cl_application класса cl_application с помощью конструктора с параметром nullptr
    ob_cl_application.build_tree_objects_4();                  //вызов      метода
build_tree_objects_4() объекта ob_cl_application
    return ob_cl_application.exec_app_4(); //вызов метода exec_app_4() объекта
ob_cl_application
}

```

## Приложение 2. Тестирование

Таблица 9 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
<pre> appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 END </pre>	<pre> Object tree appls_root   object_s1     object_s7   object_s2     object_s4     object_s6   object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 </pre>	<pre> Object tree appls_root   object_s1     object_s7   object_s2     object_s4     object_s6   object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 </pre>